Larissa Vitória Vasconcelos Sousa

Matrícula: 519221

---

1) Atividade teórica

a) (Exercícios - revisão de IA): Resolver as questões abaixo do Livro INTELIGÊNCIA Artificial (Norvig, Peter. Inteligência Artificial).

- Capítulo 1: 1.1, 1.7, 1.10, 1.11, 1.12, 1.14;
- Capítulo 2: 2.1 ao 2.13;
- Capítulo 3: 3.1, 3.2, 3.3, 3.7, 3.9, 3.1

OBS.: Aqui terá somente as resoluções práticas (códigos em python)

2.7 Escreva programas de agente de pseudocódigo para os agentes baseados em objetivos e em utilidade.

```
# Agente Baseado em Objetivos:

def agente_b_obj(percepcao):
    estado = percepcao['estado']
    objetivos = percepcao['objetivos']

    acoes = []

    for posicao, sujidade in estado.items():
        if sujidade > 0:
            acoes.append(('aspirar', posicao))
        else:
            acoes.append(('navegar', posicao))

    if objetivos == 'limpar_tudo':
        acoes.append(('terminar', None))

    return acoes


# Agente Baseado em Utilidade:

def agente_b_util(percepcao):
    estado = percepcao['estado']

    def utilidade(posicao):
        return -estado[posicao]

    acoes = []

    posicoes_sujas = [posicao for posicao, sujidade in estado.items() if sujidade > 0]

    if posicoes_sujas:
        posicao_max_utilidade = max(posicoes_sujas, key=utilidade)
        acoes.append(('aspirar', posicao_max_utilidade))

    return acoes
```

2.8 Implemente um simulador de ambiente de medição de desempenho para o mundo de aspirador de pó representado na Figura 2.2 e especificado na página 38. Sua implementação deve ser modular, de forma que os sensores, os atuadores e as características do ambiente (tamanho, forma, localização da sujeira etc.) possam ser alterados com facilidade. (Nota: Para algumas opções de linguagens de programação e sistemas operacionais, já existem implementações no repositório de código on-line.)

```
import random

class AspiradorDePo:
    def __init__(self, tamanho=2):
        self.tamanho = tamanho
        self.posicao_aspirador = random.randint(0, tamanho - 1)
```

```
        self.sujeira = [False] * tamanho

    def sujar(self, posicao):
        self.sujeira[posicao] = True

    def limpar(self, posicao):
        self.sujeira[posicao] = False

    def perceber(self):
        if self.sujeira[self.posicao_aspirador]:
            return "Sujeira"
        else:
            return "Limpo"

    def aplicar_acao(self, acao):
        if acao == "Aspirar":
            return self.aspirar()
        elif acao == "Mover para a direita":
            self.mover_direita()
            return 0
        else:
            return 0

    def objetivo_alcancado(self):
        return not any(self.sujeira)

    def mover_direita(self):
        if self.posicao_aspirador < self.tamanho - 1:
            self.posicao_aspirador += 1

    def aspirar(self):
        if self.sujeira[self.posicao_aspirador]:
            self.limpar(self.posicao_aspirador)
            return 1
        else:
            return -1
```

2.9 Implemente um único agente reflexo para o ambiente de vácuo do Exercício 2.8. Execute o ambiente com esse agente para todas as configurações iniciais sujas e localizações do agente possíveis. Registre a nota de desempenho de cada configuração e a nota média global.

```
class AgenteReflexoSimples:
    def acao(self, percepcao):
        if percepcao == "Sujeira":
            return "Aspirar"
        elif percepcao == "Limpo":
            return "Mover para a direita"
        else:
            return "Nada"

def executar_ambiente(agente, ambiente):
    total_recompensa = 0
    for _ in range(10):
        ambiente.sujar(random.randint(0, ambiente.tamanho - 1))
        while True:
            percepcao = ambiente.perceber()
            acao = agente.acao(percepcao)
            recompensa = ambiente.aplicar_acao(acao)
            total_recompensa += recompensa
            if ambiente.objetivo_alcancado():
                break
    return total_recompensa / 10

notas_de_desempenho = []

for i in range(ambiente.tamanho):
    ambiente = AmbienteAspiradorDePo(tamanho=ambiente.tamanho)  # Crie um novo ambiente para cada configuração inicial
    for j in range(ambiente.tamanho):
        nota = executar_ambiente(AgenteReflexoSimples(), ambiente)
        notas_de_desempenho.append(nota)

nota_media_global = sum(notas_de_desempenho) / len(notas_de_desempenho)
print("Notas de Desempenho para Todas as Configurações:", notas_de_desempenho)
print("Nota Média Global:", nota_media_global)
```

2.10 Considere uma versão modificada do ambiente de aspirador de pó do Exercício 2.8, na qual o agente é penalizado com um ponto para cada movimento.

b) E um agente reativo com estado? Projete tal agente.

```
# Pseudocódigo para um agente reativo com estado:

class AgenteReativoComEstado:
    def __init__(self):
        self.posicao_atual = 0
        self.ambiente = [False, False]

    def acao(self):
        if self.ambiente[self.posicao_atual]:
            self.ambiente[self.posicao_atual] = False
            return "Aspirar"
        else:

            proxima_posicao_suja = self.ambiente.index(True)
            if proxima_posicao_suja > self.posicao_atual:
                self.posicao_atual += 1
                return "Mover para a direita"
            elif proxima_posicao_suja < self.posicao_atual:
                self.posicao_atual -= 1
                return "Mover para a esquerda"
            else:
                return "Nada"
```

2.11 Considere uma versão modificada do ambiente de aspirador de pó do Exercício 2.8, na qual a geografia do ambiente — extensão, limites e obstáculos — é desconhecida, como também a configuração inicial de sujeira (o agente também pode se mover Acima e Abaixo, além de Esquerda e Direita).

b) Um agente reativo simples com uma função de agente aleatório pode superar um agente reativo simples? Projete tal agente e faça a medição de seu desempenho em vários ambientes.

```
# Pseudocódigo para um agente reativo simples com função de agente aleatório:

class AgenteReativoSimplesAleatorio:
    def acao(self, percepcao):
        acoes_possiveis = ["Esquerda", "Direita", "Acima", "Abaixo", "Aspirar", "Nada"]
        return random.choice(acoes_possiveis)
```

2.13 Os ambientes de aspiradores de pó de todos os exercícios anteriores eram determinísticos. Descreva possíveis programas de agentes para cada uma das versões estocásticas listadas a seguir:

a) Lei de Murphy: durante 25% do tempo, a ação de Aspirar falha ao limpar o chão se ele está sujo e deposita sujeira no chão se ele está limpo. De que maneira seu programa de agente é afetado se o sensor de sujeira fornece a resposta errada durante 10% do tempo?

b) Crianças pequenas: em cada período de tempo, cada quadrado limpo tem uma chance de 10% de se tornar sujo. Você poderia apresentar um projeto de agente racional para esse caso?

```
# item a
class AgenteLeiDeMurphy:
    def __init__(self):
        self.ultima_acao = None

    def acao(self, percepcao):
        if percepcao == "Sujeira":
            self.ultima_acao = "Aspirar"
            if random.uniform(0, 1) <= 0.25:
                return "Nada"
            return "Aspirar"
        else:
            self.ultima_acao = "Mover para a direita"
            return "Mover para a direita"

    def sensor_de_sujo_incorreto(self):
        if self.ultima_acao == "Aspirar" and random.uniform(0, 1) <= 0.10:
            return True
        return False
```

```
# item b
class Agente_crianca_pequena:
    def __init__(self):
        self.sujeira = False

    def acao(self, percepcao):
        if percepcao == "Sujeira":
            return "Aspirar"
        elif percepcao == "Limpo":
            if self.sujeira:
                return "Mover para a direita"
            else:
                return "Nada"

    def sujeira_aleatoria(self):
        return random.random() < 0.1
```

---

2) Atividade prática - Lista 1

```
import math

def sigmoid_derivative(a, v):
    # Calculate the sigmoid function
    phi_v = 1 / (1 + math.exp(-a * v))

    # Calculate the derivative
    derivative = a * math.exp(-a * v) / (1 + math.exp(-a * v))**2

    return derivative

# Parâmetro de inclinação 'a'
a = 2.0

# Valor de 'v' na origem
v = 0.0

# Calcular a derivada na origem
derivative_at_origin = sigmoid_derivative(a, v)

print("Derivada em v = 0:", derivative_at_origin)
```